Appeared in Mobile Robots XIII, SPIE Symposium on Intelligent Systems and Advanced Manufacturing, Boston, MA, Nov. 2-5, 1998.

A Practical Autonomous Path-Planner for Turn-of-the-Century Planetary Microrovers

S. L. Laubach[a] and J. W. Burdick[b]

[a]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA 91109, USA

[b]Dept. of Mechanical Engineering, California Institute of Technology, Pasadena, CA 91125, USA

ABSTRACT

With the success of Mars Pathfinder's Sojourner rover, a new era of planetary exploration has opened, with demand for highly capable mobile robots. These robots must be able to traverse long distances over rough, unknown terrain autonomously, under severe resource constraints. Based on the authors' firsthand experience with the Mars Pathfinder mission, this paper reviews issues which are critical for successful autonomous navigation of planetary rovers. No currently proposed methodology addresses all of these issues. We next report on the "Wedgebug" algorithm, which is applicable to planetary rover navigation in $SE(2)$. The Wedgebug algorithm is complete, correct, requires minimal memory for storage of its world model, and uses only on-board sensors, which are guided by the algorithm to efficiently sense only the data needed for motion planning. The implementation of a version of Wedgebug on the Rocky7 Mars Rover prototype at the Jet Propulsion Laboratory (JPL) is described, and experimental results from operation in simulated martian terrain are presented.

Keywords: navigation, sensor based planning, Mars rover, gaze control

1. INTRODUCTION

The recent Mars Pathfinder experience vividly illlustrated the benefits of including a mobile robotic explorer on a planetary mission. Previous forays allowed scientists to explore planets remotely, via an orbiter, or were limited to a single site for study with a lander's instruments. However, the Sojourner rover, carried to Mars by the Pathfinder spacecraft, was able to roam and to place its instruments (a spectrometer and low-mounted cameras) directly on or near objects of interest. In all, the Sojourner rover ranged over an area roughly 20 meters square, conducted soil experiments in a variety of terrains, and sampled the spectra of 16 distinct targets.[1] Missions currently being planned call for new rovers to be sent to Mars at launch opportunities in 2001, 2003, and 2005, as well as a nanorover to be sent to the surface of an asteroid in 2003. Many of these missions require the rovers to operate for up to a year, compared with the 83 sols (martian days) of operation for the Sojourner rover. The rovers are also required to traverse greatly vaster distances: up to 100 m/sol, as opposed to Sojourner's 104 m/83 sols. In addition, lessons learned from Mars Pathfinder indicate a need for significantly increased rover autonomy in order to meet mission criteria, within severe constraints including limited communication opportunities with Earth, power, and computational capacity.

1.1. Motion Planning on Mars

A key advance in functionality required for planetary rovers is greater navigational autonomy. Each rover will be working in unknown, rough terrain. (The resolution expected from Mars orbiters, for example, is roughly 300 meters/pixel, with only isolated "postage stamp" regions achieving the highest resolution of 1.4 m/pixel.[2] Orbiter camera pointing limitations prohibit attempting to use these highest-resolution images for rover navigation or localisation.) Given a distant (i.e., not immediately visible by the rover's sensors) goal designated by Earth-based operators, the rover must use its sensors to navigate safely and autonomously to that goal. Rather than address all of the issues which arise in this complex problem, this paper will focus on the aspects relevant to autonomous path planning.

Useful motion planners for planetary rovers have several key characteristics: they must assume no prior knowledge of the environment, must be sensor-based, robust, complete and correct. They must also operate under severe

Other author information: (Send correspondence to S.L.L)
S.L.L.: E-mail: Sharon.Laubach@jpl.nasa.gov
J.W.B.: E-mail: jwb@robby.caltech.edu

constraints of power, computational capacity, and the high cost of flight components, which translates into limited memory available on-board the rover. Due to dead reckoning errors, slippage on rough/loose substrate, nonholonomic fine-positioning constraints, and constraints on mission time available, using rover motion to augment sensing is costly. Simultaneously, limited memory, computational capacity, power and time available all argue for minimising the amount of data sensed and processed. Thus, a practical motion planner must utilise the available sensing array in a scheme which efficiently senses only the data needed for motion planning, requires minimal memory to store salient features of the environment, and conserves rover motion.

2. RELEVANT WORK

Much of the body of work in motion planning can be divided into three major categories: "classical" path planners, heuristic planners, and "complete and correct" sensor-based motion planners. "Classical" planners assume complete knowledge of the environment, and are complete. Heuristic planners, generally based upon a set of "behaviours," can be used in unknown environments but do not guarantee the goal will ever be reached, nor that the algorithm will halt. (A more detailed discussion is presented in (Laubach98).[3]) The third category, which relies solely upon the rover's sensors and yet guarantees completeness, is most relevant to the problem of autonomous planetary motion planning.

Two distinct approaches to such planners have been explored, both of which adapt classical methods to a local sensed region. One set of methods incrementally builds "roadmaps" within the visible region, such as Choset's HGVG,[4] Rimon's adaptation of Canny's OPP,[6] and the "Tangent Bug" algorithm of Kamon, Rivlin, and Rimon.[5] The other approach springs from approximate cell decomposition, filling in a grid-based world model as more information is gathered, exemplified by Stentz' D* algorithm.[7]

The above methods have each been developed to differing degrees in their application to real systems. For example, the sensor-based version of OPP is currently strictly theoretical, owing to the difficult-to-implement nature of the sensors required. The HGVG, on the other hand, has been implemented on a mobile robot using range sensors. Choset's planner produces paths which are maximally distant from obstacles, a plus for rover safety. However, it works best in contained environments with well-defined corridors; a description not applicable to the typical martian environment (Fig. 1).

The D* algorithm and Tangent Bug both are useful in unbounded environments. In addition, they both produce "locally optimal" solutions, that is, the resultant paths are the shortest length possible given the use of solely local information. D* has in particular been implemented on a real world system (an autonomous HMMWV driven in a slag heap near Pittsburgh). However, the grid-based world model requires a significant amount of memory for storage, and the algorithm's completeness depends entirely upon the precision of its world model, which is determined by cell granularity.

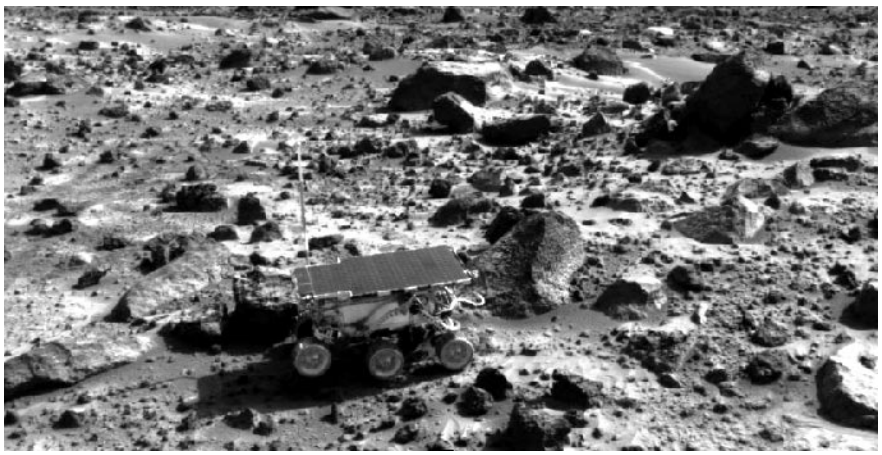

Figure 1. Typical terrain encountered on Mars by the Sojourner rover. The intrepid mobile explorer is 68cm long by 48cm wide, and stands 28cm tall.

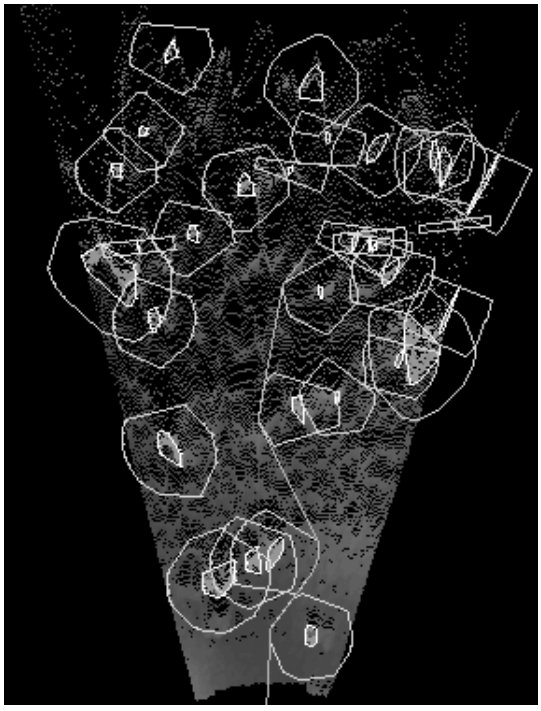

Figure 2. Rangemap of a single image from a stereo pair. This image also shows obstacles detected within the visible region, and a path generated by the implementation of the "RoverBug" algorithm on Rocky7 (see Sect. 5).

Tangent Bug provides the motivation for the work presented here. Its world model is streamlined, consisting only of sensed obstacle boundary endpoints. The planner itself consists of two "modes"— motion-to-goal, and boundary following—which interact incrementally to ensure global convergence (if the goal is reachable), and which "fail gracefully" if the goal is found to be out of reach. Thus, the algorithm is memory-efficient, fairly robust, and conserves robot motion. However, some of its assumptions do not apply to the "rover problem" of navigating in planetary terrain. For example, Tangent Bug assumes that the robot is modelled as a point, and that obstacles block both motion and sensing. In addition, Tangent Bug assumes that the robot's sensor provides an omnidirectional view.

The current scenario for a rover sensing system consists of a stereo pair of cameras mounted on a pan-able mast. Typically, these cameras have a 30° to 45° field of view (FOV), and the "visible region" connected with these sensors sweeps out roughly a wedge, with limited downrange radius R due to both viewing angle (tilt) and feature resolving ability. (See Fig. 2 for an example of data from such a sensing array.) (Camera pixels imaging features closer to the horizon (hence farther away) have a larger footprint than pixels imaging the foreground; simultaneously, obstacles further away are apparently smaller in relative size. These two properties combine to limit the range at which a stereo pair can resolve obstacles of a given height, for instance. In the case of Rocky7, discussed in Sect. 5, an obstacle 18cm tall can be resolved at roughly 6.7 m at 25° tilt.) From the discussion in Sect. 1.1, it is clear that it is important to not simply pan the sensor array and obtain an omnidirectional view at every step. Rather, the planner should be able to identify the minimal number of sensor scans needed—and which specific areas to scan—to proceed at each step, while avoiding unnecessary rover motion. Thus, we have developed the "Wedgebug" algorithm to address the shortcomings of Tangent Bug, as a step towards a more practical path planner for flight microrovers. Wedgebug is complete, correct, and relies solely upon the robot's sensors. The implementation discussed in Sect. 5 relaxes the assumption that the rover is a point robot. Perhaps most importantly, Wedgebug deals with the limited FOV of flight rovers in a manner which is efficient and minimises the need to sense and store data, using autonomous gaze control.

Section 3 presents the Wedgebug algorithm in some detail. Section 4 develops the proof of completeness for this motion planner. Section 5 describes briefly the current implementation of an extended Wedgebug on a prototype microrover at JPL, with experimental results. Section 6 contains concluding remarks.

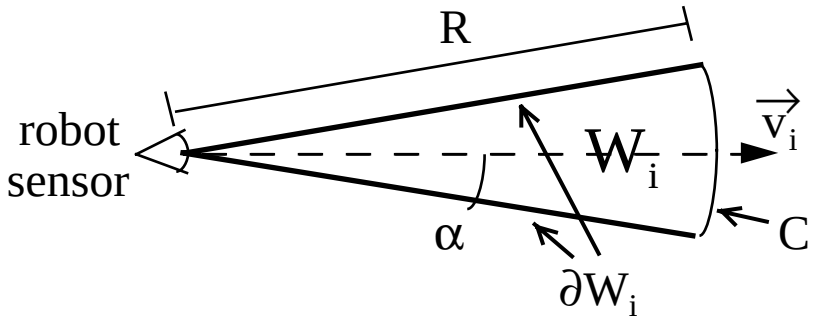

Figure 3. Anatomy of a wedge.

3. THE WEDGEBUG ALGORITHM

The basic assumptions of the Wedgebug algorithm are as follows: The rover is modelled as a point robot in a 2D binary environment (i.e., every point in the environment is either contained within an impassable obstacle, or lies in freespace). (In Sect. 5, we discuss how the implementation deals with the fact that the real robot is not a point robot.) Obstacles' boundaries block sensing as well as motion. The rover's sensing array, from position x, detects ranges within a wedge $W(x, \vec{v})$ of radius R, which sweeps out an angle 2α (> 0) and is centered on the direction $\vec{v}$. (All vectors are assumed to have unit length.) Define C as the arc boundary of $W(x, \vec{v})$ at radius R, and $\partial W(x, \vec{v})$ as the union of the two bounding rays of $W(x, \vec{v})$ (Fig. 3). We further define the "interior" of $W(x, \vec{v})$ as $\text{int}(W(x, \vec{v})) = \overline{W(x, \vec{v})} - \partial W(x, \vec{v})$ (N.B., an "interior" point may lie on C). Let $\text{d}(a, b)$ be the Euclidean distance between points a and b.

Wedgebug, like Tangent Bug, is based upon two modes which interact to ensure global convergence: *motion-to-goal* (*MtG*) and *boundary following* (*BF*). However, each mode is more finely divided into components that improve efficiency and handle the limited FOV. A high-level sketch of the operation of the Wedgebug algorithm follows: At the beginning of the path sequence, an initialisation step records the parameter $\text{d}_{\text{LEAVE}} = \text{d}(A, T)$, where A is the robot's initial position, and T is the goal. This parameter marks the largest distance the robot can stray from goal during an *MtG* segment. *MtG* is typically the dominant behaviour. It basically directs the robot to move towards the goal using a local version of the tangent graph, restricted to the visible region (Fig.4). *MtG* works roughly as follows: The robot (at position x) first senses a wedge, $W_0 = W(x, \vec{v}_0)$, where $\vec{v}_0 = \overrightarrow{xT}$ is the vector from x to the goal. (All wedges in the subsequent discussion are assumed to subsume a half-angle α.) The tangent graph consists of all line segments in freespace connecting the initial position, the goal, and all obstacle vertices, such that the segments are tangent to any obstacles they encounter.[8] Let $\text{LTG}(S)$ be the *local tangent graph* restricted to a set S, defined as the tangent graph restricted to S. The planner constructs $\text{LTG}(W_0)$. If there are no obstacles intersecting the ray $\vec{xT}$, the planner adds a node T_g to $\text{LTG}(W_0)$ at a distance R from x along $\vec{xT}$, so $\text{LTG}(W_0)$ contains a path directly towards T. The planner then searches a subgraph, $G1(W_0) = \{V \in \text{LTG}(W_0) | \, \text{d}(V, T) \leq \min(\text{d}(x, T), \text{d}_{\text{LEAVE}})\}$, for the optimal local subpath. Using the criterion discussed in Sect. 3.1, the rover may scan additional wedges as needed, and constructs the LTG in the conglomerate wedge, $\overline{W}(x)$. After executing this subpath, *MtG* begins anew. This behaviour is continued until either the goal is reached, or the robot encounters a local minimum in $\text{d}(x, T)$, which corresponds to a blocking obstacle. In the latter case, the planner switches to *BF*. The objective of this mode is to skirt the boundary of the *blocking obstacle* (the obstacle whose boundary contains the local minimum), still calculating $\text{LTG}(W_0)$, until one of two events occur: either the robot completes a loop, in which case the goal is unreachable and the algorithm halts; or $\text{LTG}(W_0)$ contains a new subpath toward the goal. The next two sections describe the *MtG* and *BF* modes in more detail.

3.1. Motion-to-Goal

During *MtG*, the robot moves toward a point (fixed for each step), called the *focus point*, F (Fig 4). This point serves as the goal for each *MtG* step. Its position within the robot's FOV also determines whether additional wedge views are needed. Initially, $F = \{V \in G1(\overline{W}(x)) | \, \text{d}(V, T) \leq \text{d}(V', T), \forall V' \in G1(\overline{W}(x))\}$. That is, since $\text{LTG}(\overline{W}(x))$ reduces in this environment to simple edges connecting the robot to the sensed obstacle boundary endpoints, F simply marks the direction for the robot to travel during this step to minimise its distance to the goal.

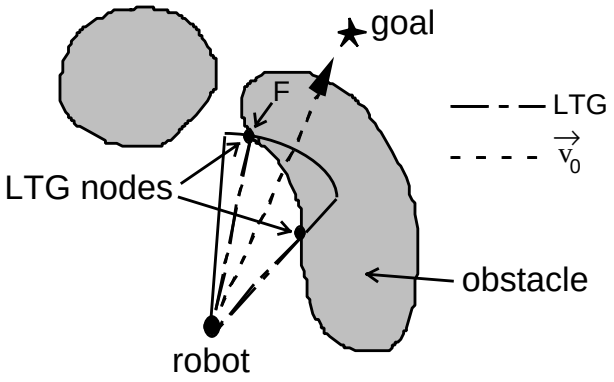

Figure 4. LTG calculated within $W(x, \vec{v}_0)$.

If $F \in \text{int}(W(x, \vec{v}_0))$, the rover simply executes the subpath to F, and starts the next *MtG* step. (N.B., for purposes of the proof to be given later, the robot never lies directly on an obstacle boundary ∂O, but rather remains a distance ε away.) We call this case a *direct* *MtG* segment, where the robot moves toward the goal through freespace. (This is the case illustrated in Fig. 4.)

If, on the other hand, $F \in \partial W(x, \vec{v}_0)$, the planner must inspect the tangent to ∂O at F, $\vec{t}_F$, to see whether the robot will be "sliding around" the blocking obstacle, or if it has possibly encountered a local minimum in $\text{d}(\cdot, T)$. If $\vec{t}_F \cdot \overrightarrow{xT} \leq 0$, the robot would need to increase its distance from the goal to skirt the obstacle on the subsequent step. So, if allowed, the planner re-searches $G1(W_0)$ for the next optimal subpath, disallowing the rejected position for F, and tests the new $\vec{t}_F$. (By the triangle inequality, if $F \in \partial\overline{W}$, then the new F must lie on the opposing bounding ray of $\overline{W}$.) Changing F is not allowed if (1) F has already been changed once at x, or (2) the change would violate the *detour condition*. This condition prevents the robot from oscillating between two directions, resulting in an unduly lengthy path, unless a clear advantage is gained by changing F. The *detour condition* states that the algorithm must track F, even as it "slides" around ∂O, and may not switch this point to a rival point y unless $\text{d}(y, T) < \text{d}(F, T)$ and $\text{d}(F, T) - \text{d}(y, T) > \text{d}_{thresh} > 0$. If the new $\vec{t}_F \cdot \overrightarrow{xT} \leq 0$ (or F cannot be changed), the robot has encountered a local minimum in $\text{d}(\cdot, T)$. Thus, the planner switches to *BF*(described in Sect. 3.2).

In the case that $F \in \partial W(x, \overrightarrow{xT})$, but $\vec{t}_F \cdot \overrightarrow{xT} > 0$, the robot must "slide around" the obstacle while progressing toward T. Unfortunately, being close to an obstacle restricts the robot's already-limited view and can result in tiny incremental steps. Thus, in order to efficiently acquire data from the robot's current position and to avoid as much inefficient motion as possible, we add a submode of *MtG*, called "virtual *MtG*". The object of "virtual *MtG*" is to sense additional wedges in the direction the robot will "slide around" the obstacle, and to generate a local shortcut in the robot path (Fig. 5).

"Virtual *MtG*" mode directs the sensing array to pan towards F (defining this direction of rotation *positive*), and to sense the wedge $W_1 = W(x, \vec{v}_1)$, where $\angle(\overrightarrow{xT}, \vec{v}_k) = 2k\alpha$ (that is, W_1 abuts W_0 at F). Let $\overline{W} = W_0 \cup W_1$ (in general, at each position x, $\overline{W}(x) = \bigcup_{sensed} W_i(x)$). The planner computes $G1(\overline{W})$, and finds the new focus point F. Let $\partial\overline{W}^+$ be the bounding ray $\vec{r}$ such that $\angle(\overrightarrow{xT}, \vec{r}) > 0$ (i.e., the edge of $\overline{W}$ in the positive direction). If $F \in \partial\overline{W}^+$, "virtual *MtG*" is repeated. This mode ends if one of three conditions is met:

1. $F \in \text{int}(\overline{W})$, in which case the robot has found a suitable shortcut. The robot executes the subpath to F, and begins a new *MtG* iteration.

2. $\angle(\overline{xT}, \partial\overline{W}^+) \geq \pi/2$, which means that the rover is sensing part of a region not useful for *MtG*, since G1 contains only nodes closer to T than the robot's current position.

3. $\vec{t}_F \cdot \overrightarrow{xT} \leq 0$, which indicates that the obstacle boundary is curving back toward x, that is, the robot can no longer "virtually slide" in this direction without losing ground.

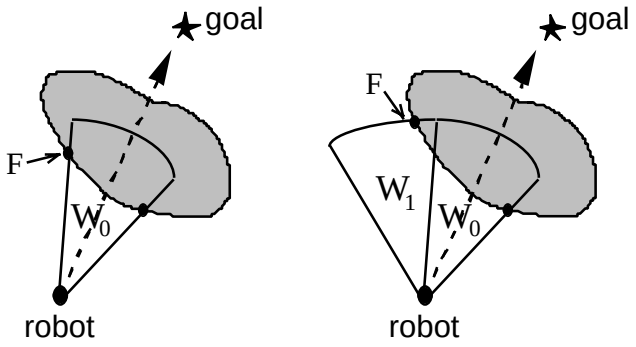

Figure 5. "Virtual *MtG*." The figure on the left depicts the first part of an *MtG* step. The nodes of LTG(W_0) are marked. F satisfies the conditions for "virtual *MtG*," so the robot scans W_1 (right). Now, $F \in \text{int}(W_0 \cup W_1)$, so "virtual *MtG*" ends.

In fact, (2) $\implies$ (3). In these cases, if allowed, the robot changes F as discussed above, and attempts "virtual *MtG*" again. If the second attempt fails, the robot has encountered a local minimum in $\text{d}(\cdot, T)$, and the planner switches to *BF*.

3.2. Boundary Following

The basic idea of *BF* is to skirt the *blocking obstacle* until progress can be made once more toward the goal. As with *MtG*, *BF* is split into two submodes. "Normal *BF*" uses two wedge views, one toward the goal and one in the direction of travel around the obstacle boundary, to determine whether a clear path towards the goal exists while the robot circumnavigates the obstacle. Immediately after a switch from *MtG* to *BF*, however, the robot must determine its direction of travel around ∂O_b, the blocking obstacle. "Virtual *BF*" is used to take full advantage of the information which can be gleaned at the current distance from the obstacle (arguably more than from a closer range), to choose this direction efficiently. (The primary motivation for "virtual *BF*" is the idea that it is less costly for the robot to swivel its sensors than for the robot to actually move.) In essence, the robot will swing its sensor array back and forth in a prescribed manner, to search for the "best" place to move and begin "normal *BF*" (Fig. 6).

More precisely, the robot initially scans the wedge $W_1 = W(x, \vec{v}_1)$, where in this case the positive direction is chosen by comparing the tangents to ∂O_b at the intersection with ∂W_0; that is, if $\vec{t}_l, \vec{t}_r$ are the two tangents (at e_l and e_r, respectively), if $\vec{t}_l \cdot \vec{v}_0 \geq \vec{t}_r \cdot \vec{v}_0$, then $\angle(\vec{v}_0, \overrightarrow{xe_l}) > 0$. As before, let $\overline{W} = W_0 \cup W_1$. The planner computes LTG($\overline{W}$) If $\exists$ a node $V \in \text{LTG}(\overline{W})$ such that $V \in \text{int}(\overline{W})$, the robot moves to V and begins "normal *BF*", first recording two features: d_{reach}, the closest point to T encountered so far on ∂O_b, and $V_{loop} = \partial \overline{W}^- \cap \partial O_b$. If there is

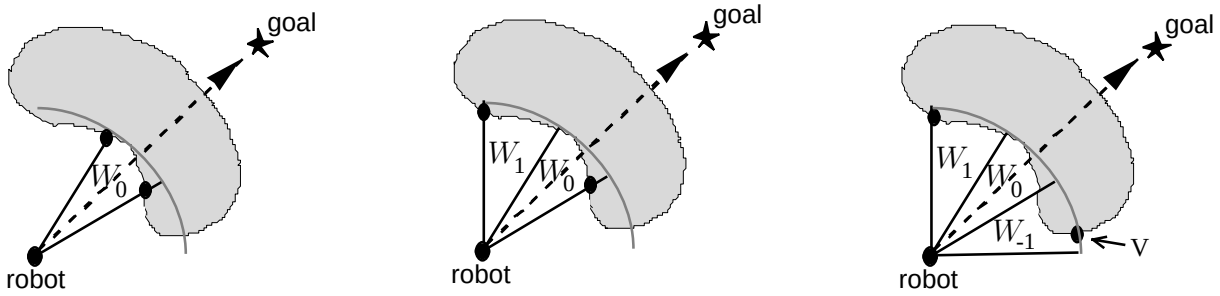

Figure 6. "Virtual *BF*." The figure on the left depicts the first part of a "virtual *BF*" step. The nodes of LTG(W_0) are marked with black circles. Since $\nexists V \in \text{int}(\text{LTG}(W_0))$, the robot scans W_1 (center). Again, $\nexists V \in \text{int}(\text{LTG}(W_0 \cup W_1))$, so the robot scans W_{-1} (right). Now, $V \in \text{int}(W_0 \cup W_1 \cup W_{-1})$, so "virtual *BF*" ends.

no such node V, the planner directs the sensing array to scan $W_{-1} = W(x, \vec{v}_{-1})$, constructs $\overline{W} = W_0 \cup W_1 \cup W_{-1}$, and searches the freshly expanded LTG($\overline{W}$). In this manner, the robot scans back and forth until a suitable node is found, then travels there to begin "normal *BF*."

"Virtual *BF*" ends when one of three events are detected:

1. $\exists V \in \text{LTG}(\overline{W}) \cap \partial O_b$ such that $V \in \text{int}(\overline{W})$. The robot moves to V, and begins normal *BF*.
2. The latest scanned wedge overlaps a previously scanned region (i.e., $|\angle(\vec{v}_0, \vec{v}_{last})| > \pi$). In this case, the robot is trapped by an encircling obstacle, and the algorithm halts.
3. $\exists V \in \text{LTG}(\overline{W})$ with $V \in \text{int}(\overline{W})$, but $V \notin \partial O_{b\ sensed}$. In this case, we call V a "framing point," since it "frames" the sensed extent of ∂O_b. The robot scans once more in the opposing direction, and then no matter the outcome, "virtual *BF*" ends. If a node as in item 1 is found, the robot moves there and begins normal *BF*. Otherwise, the rover moves to the point on ∂O_b just before the "framing node" (if there are two, V_l and V_r, the rover moves to V_l iff $|\angle(\vec{v}_0, \overrightarrow{xV_l})| > |\angle(\vec{v}_0, \overrightarrow{xV_r})|$). At this point, the rover begins normal *BF*.

In normal *BF*, at the start of each step, the robot senses W_0, and searches $G1(W_0)$. *BF* exits here if: (1) $T \in W_0$, in which case the robot moves to T and the algorithm is done, or (2) $\exists V \in G1(W_0)$ such that $\text{d}(V, T) < \text{d}_{reach}$, the *leaving condition*, in which case the planner resets d_{LEAVE} to $\text{d}(V, T)$, and begins a new *MtG* segment. If neither of these conditions hold, the planner computes $\vec{t}_x$, the tangent to ∂O_b at x, and directs the sensing array to scan $W(x, \vec{t}_x)$. If $V_{loop} \in W(x, \vec{t}_x)$, and $V_{loop} \in$ the connected portion of ∂O_b containing x, the robot has executed a loop—therefore, the goal is unreachable, and the algorithm halts. Otherwise, the planner computes $V \in \partial O_b \cap \text{LTG}(W(x, \vec{t}_x))$ such that $\text{d}(x, V) > \text{d}(x, V')\ \forall V' \in \partial O_b \cap \text{LTG}(W(x, \vec{t}_x))$. The robot records d_{reach}, executes this subpath, then begins a new *BF* step.

The Wedgebug algorithm thus deals with the limited FOV of the robot in an efficient manner. The "virtual" submodes both take advantage of the lower cost of panning the sensor array over actual motion, while minimising the number of views required at each step.

4. SKETCH OF PROOF OF CONVERGENCE

The proof of convergence of the Wedgebug algorithm is analagous to the Tangent Bug convergence proof.[5] The sketch of the proof is as follows: Each robot motion can be characterised as a particular type of motion segment. In turn, each type of segment can be shown to have finite length. Following Kamon, Rivlin, and Rimon, it can be shown that there are a finite number of each type of segment, and thus the path terminates after finite length. Since the proofs for the other types of motion segments are analagous, we will detail here only the proof that *BF* segments have finite length.

Define the points S_i to be the points where the planner switches from *MtG* to *BF*; M_i the local minimum point associated with S_i (i.e. the point $\overrightarrow{S_iT} \cap \partial O_i$); L_i the point where *BF* leaving condition is met on obstacle i (switch point from *BF* to *MtG*); and finally P_i, marker to detect loop on obstacle i, in *BF*($P_i = V_{loop}$). Then, there are two types of *BF* segments: $[S_i, L_i]$, and $[S_i, P_i]$.

LEMMA 4.1. *BF segments are finite length.*

Proof. Two cases: a) $[S_i, L_i]$. In the usual case, this segment can be considered a shortcut, compared to the path which would be taken by a robot with contact sensors executing the Tangent Bug algorithm. The "contact sensor equivalent" path consists of two pieces: $[S_i, M_i]$, and $[M_i, L_i]$, where M_i is the local minimum (along the line $\overleftrightarrow{S_iT}$) associated with S_i. Let N designate the point where the robot actually touches ∂O_i during this *BF* segment. By the triangle inequality,

$$\text{meas}([S_i, L_i]) = \text{d}(S_i, N) + \text{meas}([N, L_i]) \leq \text{d}(S_i, M_i) + \text{meas}([M_i, N]) + \text{meas}([N, L_i]).$$

We have that

$$\text{d}(S_i, M_i) \leq \text{d}(S_i, T) \leq \text{d}(\text{d}_{\text{LEAVE}}, T) \text{ (using the appropriate value for } \text{d}_{\text{LEAVE}}) \ \leq \text{d}(A, T) < \infty \text{ (by assumption).}$$

Figure 7. The Rocky7 Prototype Microrover, developed at JPL to test technologies for future missions. It is pictured here in the JPL MarsYard, an outdoor testing arena featuring simulated martian terrain.

Also, since M_i, N, and L_i all lie on ∂Oi, and the segments $\overline{M_iN}$ and $\overline{NL_i}$ do not overlap; and since the obstacle perimeter, meas(∂Oi), is finite, we have

$$\text{meas}([M_i, N]) + \text{meas}([N, L_i]) = \text{meas}([M_i, L_i]) \leq \text{meas}(\partial Oi) < \infty.$$

Thus, this segment is (crudely) bounded by

$$\begin{aligned}\text{meas}([S_i, L_i]) &\leq \text{d}(S_i, M_i) + \text{meas}([M_i, N]) + \text{meas}([N, L_i]) \leq \text{d}(S_i, M_i) + \text{meas}([M_i, L_i]) \\ &\leq \text{d}(S_i, T) + \text{meas}(\partial Oi) \leq \text{d}(\text{d}_{\text{LEAVE}}, T) + \text{meas}(\partial Oi) < \infty.\end{aligned}$$

(Of course, we also have that $\text{d}(S_i, N) \leq R$, where R is the sensing range, and $R < \infty$ by definition.)

b) $[S_i, P_i]$. Similarly, $\text{meas}([S_i, P_i]) \leq \text{d}(S_i, M_i) + \text{meas}(\partial Oi) < \infty$. □

5. IMPLEMENTATION AND RESULTS

An extended version of the Wedgebug algorithm, called "RoverBug," has been implemented on the JPL Rocky7 prototype microrover (Fig 7), a research vehicle designed to test technologies for future missions.[9] The vehicle is roughly the same size as the Sojourner rover, with a few important differences which will come into play in future rovers. (Refer to (Laubach98),[3] (Volpe96)[9] for a fuller description.) Like Sojourner, Rocky7's mobility system is a rocker-bogie suspension, capable of surmounting obstacles $1\frac{1}{2}$ wheel diameters tall (1 wheel diameter = 13 cm). However, Rocky7 boasts three stereo pairs of cameras for navigation (two body mounted, and one on a deployable 1.2m mast) as opposed to Sojourner's body-mounted laser striping system. The rover uses a photovoltaic cell-based sun sensor for absolute heading measurement. In addition, the rover software features a recently-developed localisation algorithm utilising mast imagery to aid in dead-reckoning.[10]

Although the Wedgebug algorithm is an important step, it still does not quite capture the complexities of the real world. For instance, the rover is not a point robot; a problem addressed in the "RoverBug" implementation by calculating the obstacles' "silhouettes": the smallest polygon bounding the projection of each $SE(2)$ obstacle onto $\Re^2$. Another issue is the fact that the mast imagery can "see over" many obstacles: the resulting visibility polygon is not a star-shaped set, and the LTG is much richer than in the development in Sect. 3. Also, the mast is limited in its ability to sense obstacles within 1 m of the rover, since the obstacle detection algorithm searches for steps

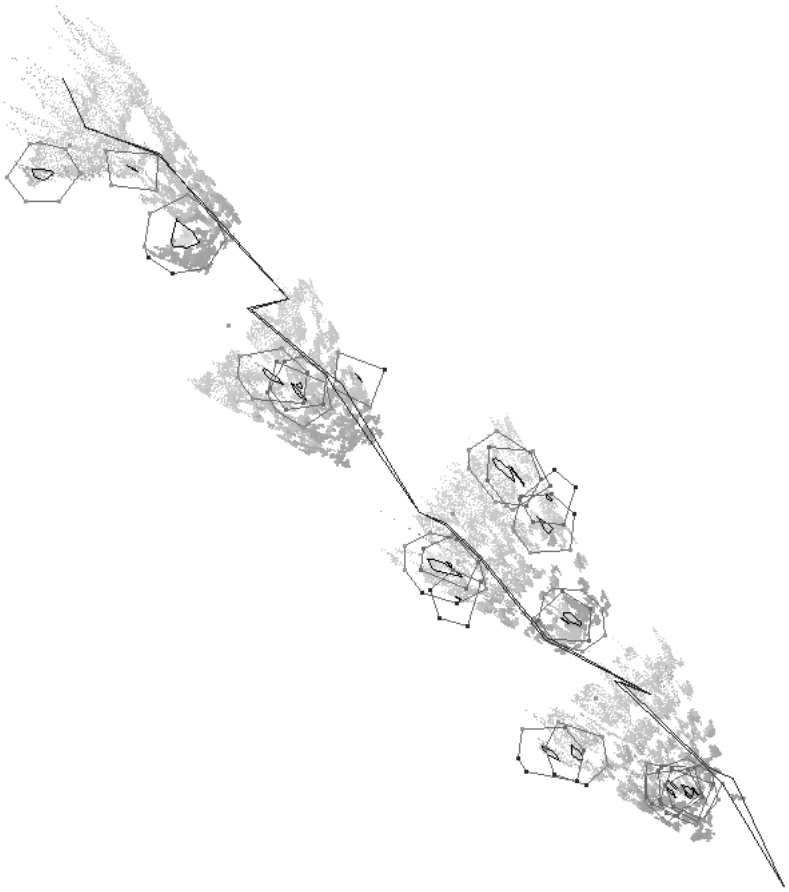

Figure 8. Results from a multi-step run in the JPL MarsYard. The path begins in the lower right corner of the image, toward a goal approx. 21 m distant in the upper left. Each wedge depicts a rangemap produced from mast imagery, and extends roughly 5 m from the imaging position. The obstacles are marked by a black convex hull, and a grey silhouette. Each subpath ends with an apparent "jag" in the path; these are not in fact motions, but rather the result of the localisation procedure run at the conclusion of each step. The second line echoing the path is the rover's telemetry for the run.

in elevation, not easy to detect while looking straight down on the tops of rocks. Thus, care must be taken while executing the subpaths.

In brief, the scenario is as follows: The rover is situated in unknown, rough terrain. The remote human operator designates a goal, using imagery from, for example, either the rover or an orbiter as a guide. This action sets in motion the autonomous planner. The planner begins by directing the mast to image towards the goal. Software on-board produces a rangemap from the stereo imagery, and detects "obstacle points"—pixels in the range image determined to be part of an obstacle—using a simple height/slope model within the range image. Additional functions segment the detected obstacle points into discrete 2D connected obstacles, and computes the obstacles' convex hulls. The planner, which uses a version of the theory described above, then computes the obstacles' silhouettes, and searches the resulting LTG using a modified A* algorithm[11] to produce the first subpath. The planner directs the mast to look in the appropriate direction(s), and incrementally builds the local subpath. Before and after the execution of each subpath, the localisation procedure is called, to verify the rover's new position. Then, the mast is directed to image toward the goal, and the process repeats until the goal is reached. "RoverBug" is amenable to varying levels of autonomy, from single-step paths under tight operator control, to full multi-step autonomy as described here.

The implementation so far has been tested extensively in the JPL MarsYard, as well as in natural arroyo terrain. Figure 8 shows the results of one typical run in the MarsYard. The goal was approximately 21 m distant from

the initial position, and R for each wedge (the radius of useable data) was 5 m*. The rover's initial position was in the lower right corner. As in Fig.2, the convex hulls and silhouettes were computed within each wedge view, and a subpath generated as described above. This subpath was executed before the rover redeployed its mast for localisation. The results of the localisation procedure appear in the figure as "jags" in the path at the end of each subpath. These features are not actual motions, but rather updated final positions. The rover was next directed by the planner to image toward the goal from its new position. This process was repeated, for a total of four wedges and subpaths, until the goal was achieved. The resultant multi-step path runs from lower right to upper left.

6. SUMMARY AND CONCLUSIONS

The requirements for autonomous flight rovers for planetary exploration provide compelling motivation for work in streamlined sensor-based motion planning. This paper addresses some of the issues related to autonomous path-planning for planetary microrovers expected to traverse hundreds of meters between uplink opportunities. The paper continues the work begun in (Laubach98)[3] to develop, implement, and test a robust, practical path planner for the Rocky7 prototype microrover. The Wedgebug algorithm is described, along with its proof of convergence. A companion paper will describe in more detail the "RoverBug" planner, the Wedgebug extension implemented on Rocky7. These planners significantly augment microrovers' autonomous navigation ability, which in turn will aid in producing successful mobile robot missions.

ACKNOWLEDGMENTS

The work described here was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. We would like to acknowledge the Long Range Science Rover team and the Mars Pathfinder Microrover Flight Experiment team, for help, inspiration, and flight experience with a rover. The authors would particularly like to thank Samad Hayati, Andrew Mishkin, Clark Olson, Rich Petras, and Todd Litwin for their invaluable assistance.

REFERENCES

1. A. Mishkin, J. Morrison, T. Nguyen, H. Stone, and B. Cooper, "Operations and autonomy of the Mars Pathfinder Microrover," in *Proc. IEEE Aerospace Conf.*, 1998.
2. Jet Propulsion Laboratory Document D-12487, *MGS Investigation Description and Science Requirement Document*, February 1995.
3. S. L. Laubach, "An autonomous path-planner implemented on the Rocky7 prototype microrover," in *Proc. IEEE Conf. Robotics Automat.*, 1998.
4. H. Choset, *Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph*. PhD thesis, California Inst. of Tech., 1996.
5. I. Kamon, E. Rivlin, and E. Rimon, "A new range-sensor based globally convergent navigation algorithm for mobile robots," tech. rep., CIS–Center of Intelligent Systems 9517, Computer Science Dept., Technion, Israel, 1995.
6. E. Rimon and J. Canny, "Construction of C-space roadmaps from local sensory data: What should the sensors look for?," in *Proc. IEEE Conf. Robotics Automat.*, 1994.
7. A. Stentz, "Optimal and efficient path planning for partially-known environments," in *Proc. IEEE Conf. Robotics Automat.*, 1994.
8. J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, 1991.
9. R. Volpe, J. Balaram, T. Ohm, and R. Ivlev, "The Rocky7 Mars Rover Prototype," in *Proc. IEEE/RSJ Conf. Intelligent Robots and Sys.*, 1996.
10. C. Olson and L. Matthies, "Maximum likelihood rover localisation by matching range maps," in *Proc. IEEE Conf. Robotics Automat.*, 1998.
11. A. Rankin, "Path planning and path execution software for an autonomous nonholonomic robot vehicle," Master's thesis, University of Florida, 1993.

*The 5 m limit on useable range data in this case was determined not only by the camera limitations discussed in Sect. 2, but also by the empirical limit on traverse distance for localisation effectiveness. The camera tilt angle was adjusted accordingly, to 29°